# CS246 Spring 2020 Project – Hydra

C. Kierstead G. Richards G. Tondello

Due Date 1: Wednesday, August 5, 5:00pm
Due Date 2: Saturday, August 15, 11:59pm

**DO NOT EVER SUBMIT TO MARMOSET WITHOUT COMPILING AND TESTING FIRST. If your final submission doesn't compile, or otherwise doesn't work, you will have nothing to show during your demo. Resist the temptation to make last-minute changes. They probably aren't worth it.**

**This project is intended to be doable by one person in two weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes two weeks' worth of work is difficult to nail down. Some students may finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.**

**Above all, MAKE SURE YOUR SUBMITTED PROGRAM RUNS. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something.**

## Hydra

In this project, you will implement a card game, Hydra. This section will describe the card game as it is played with cards, and the next section will describe how it is to be computerized. Hydra is called so because a number of active card stacks are maintained, each called a "head", and whenever a head is removed from the game, two new heads are created. The goal of each player in Hydra is to lose all of their cards by placing them on the heads, but whenever a head is removed, it is reshuffled into the offending player's deck.

To start a game of Hydra, one needs as many standard, 54-card decks (i.e., with jokers) as there are players. These decks are shuffled together, then distributed equally to each player, face down in a pile. This pile is each player's *draw pile*, and each player additionally has a *discard pile*, which starts empty, and during a turn, a *reserve*. To start the game, the first player places a single card from the top of their draw pile face up in the center of the table, creating the first "head". If the card happens to be a joker, then it is taken to have the value 2 (see the discussion of jokers below). From there, play proceeds clockwise.

In a turn of Hydra, one must draw and play as many cards as there are currently heads, unless an action ends the player's turn early. For instance, if there are three heads, the player must play three cards. To play a card, the player draws it from the top of the player's draw pile, and performs an action with it. If the player's draw pile is empty, the player must first shuffle their discard pile to form a new draw pile, then draw from that. If the player's draw and discard piles are both empty, and the player has no reserve, the player wins. There are several possible actions, and conditions under which the actions are allowed:

1. If the card's face value is strictly less than the value of the top card of any head (an ace is considered to have the value 1, and so has the lowest value), the card may be placed, face up, on that head.

2. If the card's face value is equal to the face value of the top card of any head, the card may be placed, face up, on that head, but this ends the player's turn immediately, and they draw no further cards.

3. If the top card of any head is an ace, and the played card is not, then it may be placed, face up, on that head.

4. If the player has no reserve, and there is more than one head, they may place this card into reserve. To do so, the player simply places the card, face down, in front of them.

5. If the player has a reserve, then the player may swap the played card for the reserve card.

In the first three actions, if the played card is a joker, then the player determines and announces a value for the joker while playing it, and the value announced must satisfy the requirements for the action. For instance, the player may place a joker on a 3, by announcing that the joker is now an ace, 2, or 3. In the latter case, as this would fit action 2, this also ends the player's turn. The joker retains its announced value as long as it remains on top of the head it was placed on, so in our above example, the next player must treat the joker as an ace, 2, or 3 (whichever it was announced to be), not a joker.

If the player had more cards to draw in the current turn, then after the first, third, or fourth action, they simply continue. If they are required to draw more cards, and they have more cards in their draw and/or discard pile, they draw another card; if they are required to draw more cards, but their only remaining card is the reserve, then their reserve card is added to their discard, and their turn ends; if they are out of cards entirely, they win; if they are not required to draw another card and still have cards, then their reserve, if any, is added to their discard pile, and their turn ends. After the second action, the player's reserve, if any, is added, face down, to their discard pile, and play continues with the next player, unless the current player is out of cards, in which case they win. After the fifth action, the player must play the card retrieved from the reserve, and this does not count as drawing another card: That is, if the player must play five cards (because there are five heads), swapping four times does not free them of this requirement.

A few subtleties to note:

- Aces are "universal": They may be played on any head, and if they are the top card on any head, then any card may be played on them. The only caveat is that if a player plays an ace on an ace, that ends their turn, leaving the universal card free for the next player.

- Jokers are wildcards, but are *not* universal in the sense that aces are, since their value is restricted by announcing it. A joker can be played as an ace, of course, in which case it is universal.

- A player who's drawn a card of equal value to an existing head may end their turn, removing any further risk that turn, but also keeping all of the cards they would otherwise have been able to play that turn. Deciding when it's best to end one's turn involves strategy.

- A player cannot win in a turn in which they've used the reserve, because the reserve card itself is not played.

- Otherwise, a player can win at any point during their turn.

If none of the first four actions apply, and the player does not wish to swap with their reserve, the player must perform the following sequence of actions, called "cutting off" a head:

- Place the played card into their discard pile, face down.

- Place their reserve card, if any, into their discard pile, face down.

- Move every card in the oldest head, face down, into their discard pile.

- With the top two cards from the player's draw pile (reshuffling the discard pile if necessary), create two new heads by placing the two cards, face up, as two new piles. Any jokers placed in this manner are given the value 2.

Generally speaking, this action will give them more cards, rendering them further away from victory.

After that, play continues with the next player. Note however that it is possible to "win by losing": If the player's last card does not match any action, they will be forced to cut off a head. But, if the oldest

head consists of only a single card, and the player had no reserve at the time, then the two cards they play to create two new heads are their last two cards. They will thus be rendered without any cards, and win!

Play continues until a player wins.

Hydra isn't an especially strategic game, but what strategy there is comes from a combination of careful use of the reserve and card counting. For instance, if a player has determined that a particular head is particularly rich in low cards, or universal aces, they may play quite boldly, confident that while they may end up cutting off a head and thus gaining more cards, the cards they end up with will be easier to play in the future than the cards they currently have. They may even use the reserve to hold a high-valued card, then swap with the reserve to intentionally "lose", enriching their future draw pile with easily-played low cards. Alternatively, if they draw an ace, they would be wise to put it in their reserve, allowing them to play recklessly, since they can always swap for their ace and render their situation more playable.

## Computer Hydra

**Question:** What sort of design pattern should you use to structure your game classes so that changing the interface or changing the game rules would have as little impact on the code as possible? Explain how your classes fit this framework.

If you've carefully read the rules of Hydra, you've probably noticed that the suit is irrelevant. Nonetheless, you must remember the suit. Each card is represented by the concatenation of its value (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) with a character for its suit (S, H, C, D), e.g. 10D or KS. A joker in the hand is represented by the string "`Joker`", but a joker that's been played and assigned a value is represented by that value, with J as the "suit", e.g. 3J. **Question:** Jokers have a different behaviour from any other card. How should you structure your card type to support this without special-casing jokers everywhere they are used?

When the game begins, Computer Hydra should prompt "`How many players?`", input a number greater than 1, then start a game with that many players. Internally, it should then shuffle together the decks, distribute the cards evenly among the players, and even perform the first player's move, removing their first card and starting a head.

After that, every time the program interacts with a player, it should first print the a blank line, the list of heads, the list of players, and then the interactive prompt.

**Question:** If you want to allow computer players, in addition to human players, how might you structure your classes? Consider that different types of computer players might also have differing play strategies, and that strategies might change as the game progresses i.e. dynamically during the play of the game. How would that affect your structure?

**Question:** If a human player wanted to stop playing, but the other players wished to continue, it would be reasonable to replace them with a computer player. How might you structure your classes to allow an easy transfer of the information associated with the human player to the computer player?

The list of heads consists of the line "`Heads:`", one line per head, then a blank line. Each head is printed in the following pattern: "*a*`: ` *b* ` (`*c*`)`" *a* is the number of the head. The first head in the game has number 1, but when a head is removed from the game, its number is out of the game forever, so the first head printed won't usually be number 1. *b* is the value of the top card on the head. *c* is the total number of cards in the head, including the top card. For example:

```
Heads:
2: A (12)
3: 9 (3)
4: 10 (6)
```

The list of players consists of the line "`Players:`", one line per player, then a blank line. There are two forms of the player line: One if the player is the current player, and one if they are not.

If they are not the current player, the line follows this pattern: "`Player` *a*`: ` *b* ` (`*c* `draw, ` *d* `discard)`". *a* is the player's number, starting from 1. *b* is the sum of the number of cards in their draw and discard piles. *c* is the number of cards in their draw pile. *d* is the number of cards in their discard pile.

If they are the current player, the line follows this pattern: "`Player` *a*`: ` *b* ` (`*c* `draw, ` *d* `discard) + 1 in hand, ` *e* `remaining, ` *f* `in reserve`". *a*, *b*, *c*, and *d* are as above. *e* is the number of cards they still must

play in their current turn, after the one in their hand. $f$ is either 1 or 0, indicating whether they do or do not have a reserve. Note that if the player is playing, they have exactly one card in their hand by definition.

There are two interactive prompts for players:

1. "`Player` $a$`, it is your turn.`". $a$ is the player's number, starting from 1. The player must enter any line (typically a blank line) to continue play. Note that when this prompt is displayed, it is no one's turn, so all of the player lines should be in the form taken when it is not the player's turn.

2. "`Player` $a$`, you are holding a` $b$`. Your move?`". $a$ is the player's number, starting from 1. $b$ is the value of the card in the player's hand, or "`Joker`" if the card is a joker. The player is expected to enter their move, as a number.

The move is either the number of an active head, or `0`. `0` refers to either placing the current card in reserve, or swapping the current card for the reserve card, depending on whether the player currently has a reserve card. If the player selects a head, but there is no valid move with that head, the game simply ignores the invalid move and prompts again. The same is true for a number greater than 0 but less than the first current head, or greater than the number of the last current head. If the card being placed is a joker, and a head has been selected, a further prompt requests "`Joker value?`". The user must enter one of A, J, Q, K, or a number between 2 and 10, representing the value. If the entered value is not valid for the given play, then the play is ignored, and the player's turn repeats. Note that cutting off a head is done by inputting the number of the first head, and this action is only valid when there is nowhere else the player can play their current card. In any case, the game updates the state of the cards according to the rules of Hydra, and either continues play or declares the player the winner.

The game is over when a player is out of cards, which can happen at several points during their turn, outlined in the rules above. When a player wins, the program prints "`Player` $a$ `wins!`", where $a$ is the winning player's number, and then quits.

## Example

This is an example game of Hydra, including only a few interesting snippets. The *[...]* indicates where the rest of the game has been excluded.

```
How many players?
2

Heads:
1: 3H (1)

Players:
Player 1: 53 (53 draw, 0 discard)
Player 2: 54 (54 draw, 0 discard)

Player 2, it is your turn.


Heads:
1: 3H (1)

Players:
Player 1: 53 (53 draw, 0 discard)
Player 2: 53 (53 draw, 0 discard) + 1 in hand, 0 remaining, 0 in reserve

Player 2, you are holding a 5H. Your move?
1
```

```
Heads:
2: KD (1)
3: 2D (1)

Players:
Player 1: 53 (53 draw, 0 discard)
Player 2: 53 (51 draw, 2 discard)

Player 1, it is your turn.
```

*[...]*

```
Player 1, you are holding a Joker. Your move?
13
Joker value?
A

Heads:
9: 4D (6)
10: 4C (6)
11: 3S (2)
12: 10H (8)
13: AJ (4)
14: AC (3)
15: 7C (6)
16: 5C (2)
17: 3H (1)

Players:
Player 1: 47 (12 draw, 35 discard) + 1 in hand, 2 remaining, 1 in reserve
Player 2: 21 (8 draw, 13 discard)

Player 1, you are holding a QC. Your move?
```

*[...]*

```
Heads:
13: 3H (10)
14: 2H (12)
15: 9S (13)
16: 2C (11)
17: 8C (5)
18: KD (5)
19: 8H (3)
20: 8C (8)
21: 6H (8)
22: 9C (1)
23: 7H (2)
24: 7C (1)
25: 4H (4)

Players:
Player 1: 23 (6 draw, 17 discard)
Player 2: 1 (1 draw, 0 discard) + 1 in hand, 12 remaining, 0 in reserve

Player 2, you are holding a 7D. Your move?
```

```
17

Heads:
13: 3H (10)
14: 2H (12)
15: 9S (13)
16: 2C (11)
17: 7D (6)
18: KD (5)
19: 8H (3)
20: 8C (8)
21: 6H (8)
22: 9C (1)
23: 7H (2)
24: 7C (1)
25: 4H (4)

Players:
Player 1: 23 (6 draw, 17 discard)
Player 2: 0 (0 draw, 0 discard) + 1 in hand, 11 remaining, 0 in reserve

Player 2, you are holding a 10H. Your move?
18
Player 2 wins!
```

### Testing Mode

Generally, the Hydra game should take no command line arguments. But, in order to facilitate testing, a command line argument `-testing` must be accepted, which enables testing mode. In testing mode, every time the game is about to request a move (the prompt `Player` *a*`, you are holding a`...), it will first allow you to choose the value and suit of the card you're holding. It will ask `Card value?`, and expect A, J, Q, K, 2-10, or Joker in response. If the value was not Joker, it will then ask `Suit?`, and expect S, H, C, or D in response. In addition, it will ask this during the normally non-interactive card selections of the first move (player 1 places a card as the first head), and cutting off heads. If the response to either of these questions is invalid, it will repeat the question until it receives a valid response. In this way, the game becomes testable, since the only source of randomness is the order of the cards.

### Other Options

You may find it useful to implement another optional argument that represents a seed for your random generation (so that you can have reproducible results). This is **not** required.

## Grading

Your project will be graded as follows:

| | | |
|---|---|---|
| Correctness and Completeness | 60% | Does it work? Does it implement all of the requirements? |
| Documentation | 20% | Plan of attack; Final design document. |
| Design | 20% | UML; good use of separate compilation, good object-oriented practice; is it well-structured, or is it one giant function? |

Even if your program doesn't work, you can still earn marks associated with the Documentation and Design components of the project.

## If Things Go Well

If you complete the entire project, you can earn up to 10% extra credit for implementing extra features. You should provide some way of playing the game with and without enhancements, as this will allow project assessors to accurately determine that you met all of the base requirements. Recompiling is **not** permitted.

The following is a list of possible enhancements that you could develop. However, don't attempt to implement this until you've got the base game working.

- Grammar. The game says you're holding "a A" or "a 8" instead of "an".
- House rules. Sometimes Hydra is played with red cards going down (as usual) but black cards going up. Sometimes Hydra is played allowing you to cut off a head even if there is a valid play. Or, you could invent your own house rules.
- Better display. Even with just ASCII, the "table" could be drawn more nicely than a list of heads. Or if you're feeling bold, you could implement a GUI.

To earn significant credit, enhancements must be algorithmically difficult, or solve an interesting problem in object-oriented design. Trivial enhancements will not earn a significant fraction of the available marks.

## Due Dates

**Due Date 1:** Due on due date 1 is your plan of attack, `plan.pdf`, and initial UML diagram, `uml.pdf`, for your implementation of Hydra on Maromset, `CS246_PROJECT`.
**Due Date 2:** Due on due date 2 is your actual implementation of Hydra. All .h, .cc and any other files needed for your project to compile and run should be included in `hydra.zip`. The zip file must contain a suitable `Makefile` such that typing `make` will compile your code and produce an executable named `hydra`. In addition, upload `demo.zip`, which contains `demo.pdf` and all necessary files to run your demo (such as saved input, either with a fixed seed or in testing mode, but not the program executable, which will be compiled from the files submitted in `hydra.zip`), `uml-final.pdf` and `design.pdf` to the appropriate place on Marmoset, `CS246_PROJECT`.

If you have implemented any bonus features, make sure that your submitted demo plan includes a list of all of the described bonus features, and how to run the game to see the effect of the bonus features.

See **project_guidelines-online.pdf** for instructions about what should be included in your plan of attack and final design document.

## A Note on Random Generation

To complete this project, you will require the random (or rather, pseudo-random) generation of numbers. You have several options available to you.

In `<algorithm>`, there is the `shuffle` function, which requires a random engine. Documentation on `shuffle` usually describes how to choose and use random seeds.

Alternatively, in `<cstdlib>`, there are functions **rand** and **srand**, which generate a random number and seed the random generator respectively (typically, seeded with **time()** from `<ctime>`).